Author = Song Jian

#### 阅读论文： DEEP GRAPH INFOMAX

|  |  |  |
| --- | --- | --- |
| @article{Petar2019,  author = {Petar, Veli{\v{c}}kovi{\'{c}} and William, Fedus and Hamilton, William L. and Pietro, Li{\`{o}} and Yoshua, Bengio and Hjelm, R Devon},  pages = {1--17},  title = {{Deep graph infomax}},  year = {2019}  } | [1] V. Petar, F. William, W. L. Hamilton, L. Pietro, B. Yoshua, and R. D. Hjelm, “Deep graph infomax,” pp. 1–17, 2019.  论文代码：<https://github.com/PetarV-/DGI> | 剑桥大学  谷歌大脑 |

#### 文档由以下几个部分组成：

* 文章简述
* 理论背景
* 实验部分
* 代码部分

# 1文章简述

本篇文章的主要工作是基于DIM的启发，使用互信息最大化的LOSS替代传统随机游走式的LOSS，以无监督的方式去学习节点的嵌入向量。

## 1.1经典的方法

在对网络进行无监督地特征学习中，大多是基于“随机游走式的目标函数”的一些算法，比如Deepwalk, node2vec等，这些算法在设置目标函数时，目的都是想让在原本网络中相近的节点，在嵌入空间中，也是相近的。

## 1.2经典方法的缺陷

随机游走式目标函数的这些算法主要是根据邻接矩阵，学习出PPR(Personalized PageRank)值，从而求出每一个节点对之间的相近程度。这些算法主要会有两个限制：

* 一是过于强调节点的邻近信息，忽略了节点的结构信息；
* 二是算法对超参比较敏感。

目前为止，大多数的GNN都是以图卷积为主，经过几次卷积之后，相邻的节点已经有了相似的向量，此时随机游走式的目标函数可能就无法在提供有效的梯度指示了。

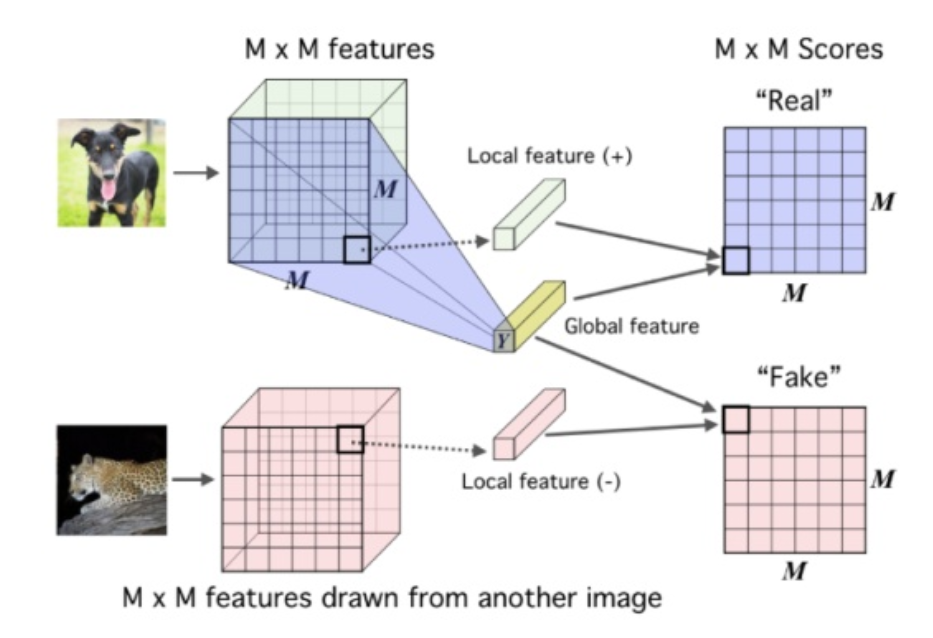
# 2理论背景

## 2.1互信息

“互信息”是信息熵的引申概念，它是对两个随机事件相关性的度量。比如说随机事件，今天北京下雨和随机变量空气湿度的相关性就很大，但是和姚明所在的休斯顿火箭队能否赢公牛队几乎无关。互信息就是用来量化度量这种相关性的。可以用以下公式表示：

其实互信息就是随机事件的不确定性或者说熵，以及在知道随机事件条件下的不确定性或者说条件熵之间的差异(摘自吴军老师的《数学之美》)。

## 2.2 DIM



Deep InfoMax (Hjelm et al. 2018) 1

Deep InfoMax通过利用图像中的局部结构来学习图像表示，如上图所示。这个方法中的对比任务是对一对图像中的全局特征和局部特征进行分类。在这里，全局特征是CNN的最终输出（平面向量，Y），局部特征是编码器中的中间层的输出（M x M特征图）。 每个局部特征图都有一个有限的感受野。 因此，要出色的完成对比任务，全局特征必须捕获来自所有不同局部区域的信息。

DIM的两个核心idea：

* 训练一个编码器，最大化输入与输出的互信息。
* 最大化局部特征与全局特征的互信息

## 2.3负采样

说到噪声对比估计，或者“负采样”，大家可能立马就想到了Word2Vec。在原理和做法上DIM跟 Word2Vec大体都一样。在Word2Vec中，也是随机采集负样本，然后通过判别器来区分两者的过程。这个过程我们通常称为“噪声对比估计”，之前也提到过，Word2Vec的噪声对比估计过程（负采样）的实际优化目标就是互信息（摘自苏剑林的公众号paperweekly）。

## 2.4 DGI METHODOLOGY

根据上面对的介绍，可以知道将DIM算法应用到网络领域上。需要解决4个问题：

* 如何得到局部表示(patch representations);
* 如何得到全局摘要(global summaries);
* 如何得到负样本的局部表示；
* 如何设计分类器D，区分正负patch-summary pairs

#### 2.4.1局部表示Patch representations

作者的目标是学习一个编码器，比如，用于表示每个节点的高阶表示。

这个编码器就是GCN模型，因为GCN可以把邻近节点的信息汇集到一个点上，因此又可以称为patch representations

#### 2.4.2全局摘要Global Summaries

全局摘要通过readout函数来得到，文中尝试了几种readout函数(比如set2vec)，发现效果最好的前实现起来最简单的就是对所有的patch representations取平均。

#### 2.4.3获取负采样的局部特征

文中采用一个腐蚀函数(corruption function） ，例如，简单来说就是，邻接矩阵不变，输入特征矩阵X进行行变换，并结合之前计算出的全局摘要来做负样本。

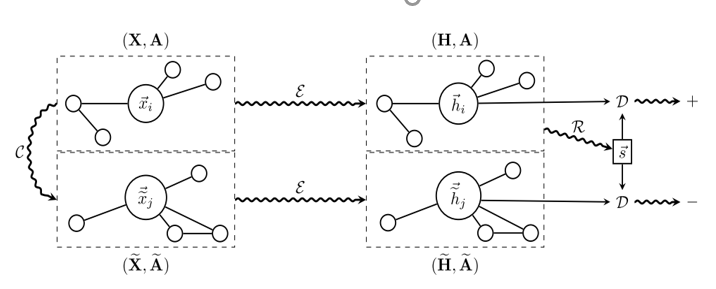
对于负采样的样本选取，单图中采用将节点的特征矩阵打乱制造负样本，多图中随机采用其他图作为负样本。

#### 2.4.4 线性分类器D

应用简单的双线性评分函数（类似于Oord等人（2018）使用的评分）对摘要-局部表示对进行评分。

#### 2.4.5 DGI模型的计算过程

1. 通过腐蚀函数C构造负样本
2. 通过图卷积得到正样本和负样本各自的
3. 定义函数得到整个图的全局摘要
4. 通过梯度下降优化下面的目标函数来更新，和𝒟的参数，优化L相当于最大化节点表示与图表示S之间的互信息。



DGI的概览 1

# 3实验部分

#### 3.1数据集

数据集：Citeseer(Transductive)，Cora(Transductive)，PubMed(Transductive)，Reddit(Inductive)，PPI(Inductive)

使用(Zhilin Yang, William W. Cohen, Ruslan Salakhutdinov, Revisiting Semi-Supervised Learning with Graph Embeddings, ICML 2016)对数据集进行划分。

以cora为例：

ind.dataset\_str.x => 训练实例的特征向量，是scipy.sparse.csr.csr\_matrix类对象，shape:(140, 1433)

ind.dataset\_str.tx => 测试实例的特征向量,shape:(1000, 1433)

ind.dataset\_str.allx => 有标签的+无无标签训练实例的特征向量，是ind.dataset\_str.x的超集，shape:(1708, 1433)

ind.dataset\_str.y => 训练实例的标签，独热编码，numpy.ndarray类的实例，是numpy.ndarray对象，shape：(140, 7)

ind.dataset\_str.ty => 测试实例的标签，独热编码，numpy.ndarray类的实例,shape:(1000, 7)

ind.dataset\_str.ally => 对应于ind.dataset\_str.allx的标签，独热编码,shape:(1708, 7)

ind.dataset\_str.graph => 图数据，collections.defaultdict类的实例，格式为 {index：[index\_of\_neighbor\_nodes]}

ind.dataset\_str.test.index => 测试实例的id，2157行

上述文件必须都用python的pickle模块存储

#### 3.2实验描述

作者分别在三类任务重应用了DGI模型分别是 Transductive，Inductive 以及多个图之间的 Inductive，主要区别在于构造负样本的方法上。

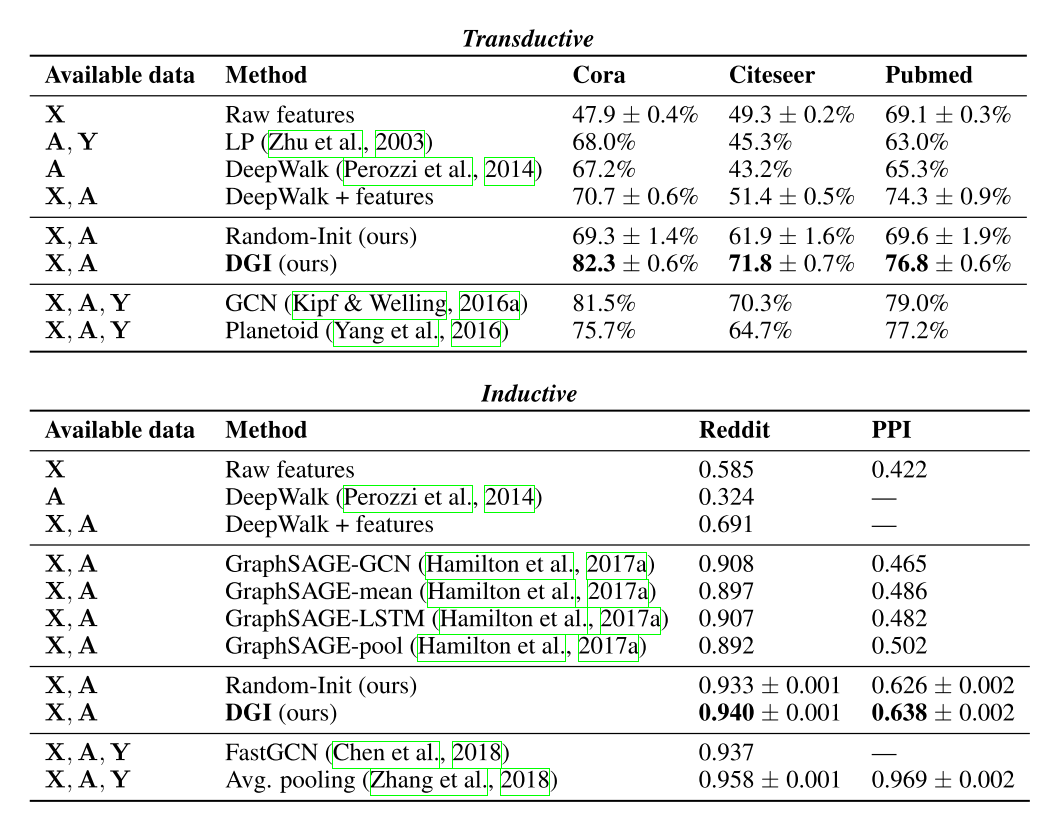
在Transductive 和Inductive 任务中，由于节点都属于同一个图，作者在构造负样本时保持邻接矩阵不变，按行打乱节点特征矩阵X，相当于交换原图中某些点的位置；

在多个图之间的 Inductive任务中，作者随机选择一个图作为负样本进行目标函数的计算。由于 DGI 在学习节点表示的过程中没有任何监督标记，因此需要另外训练分类器（如逻辑回归）得到最终的分类结果。

X：features

A：adjacency matrix

Y：labels



实验结果 1

# 4代码部分

#### 4.1gcn模块

class GCN(nn.Module):

def \_\_init\_\_(self, in\_ft, out\_ft, act, bias=True):

super(GCN, self).\_\_init\_\_()

self.fc = nn.Linear(in\_ft, out\_ft, bias=False)

self.act = nn.PReLU() if act == 'prelu' else act

if bias:

self.bias = nn.Parameter(torch.FloatTensor(out\_ft))

self.bias.data.fill\_(0.0)

else:

self.register\_parameter('bias', None)

for m in self.modules():

self.weights\_init(m)

def weights\_init(self, m):

if isinstance(m, nn.Linear):

torch.nn.init.xavier\_uniform\_(m.weight.data)

if m.bias is not None:

m.bias.data.fill\_(0.0)

# Shape of seq: (batch, nodes, features)

def forward(self, seq, adj, sparse=False):

seq\_fts = self.fc(seq)

if sparse:

out = torch.unsqueeze(torch.spmm(adj, torch.squeeze(seq\_fts, 0)), 0)

else:

out = torch.bmm(adj, seq\_fts)

if self.bias is not None:

out += self.bias

return self.act(out)

#### 4.2判别器模块

class Discriminator(nn.Module):

def \_\_init\_\_(self, n\_h):

super(Discriminator, self).\_\_init\_\_()

self.f\_k = nn.Bilinear(n\_h, n\_h, 1)

for m in self.modules():

self.weights\_init(m)

def weights\_init(self, m):

if isinstance(m, nn.Bilinear):

torch.nn.init.xavier\_uniform\_(m.weight.data)

if m.bias is not None:

m.bias.data.fill\_(0.0)

def forward(self, c, h\_pl, h\_mi, s\_bias1=None, s\_bias2=None):

c\_x = torch.unsqueeze(c, 1)

c\_x = c\_x.expand\_as(h\_pl)

sc\_1 = torch.squeeze(self.f\_k(h\_pl, c\_x), 2)

sc\_2 = torch.squeeze(self.f\_k(h\_mi, c\_x), 2)

if s\_bias1 is not None:

sc\_1 += s\_bias1

if s\_bias2 is not None:

sc\_2 += s\_bias2

logits = torch.cat((sc\_1, sc\_2), 1)

return logits

#### 4.3 DGI模块

class DGI(nn.Module):

def \_\_init\_\_(self, n\_in, n\_h, activation):

super(DGI, self).\_\_init\_\_()

self.gcn = GCN(n\_in, n\_h, activation)

# readout函数获取全局摘要

self.read = AvgReadout()

self.sigm = nn.Sigmoid()

# 判别器函数

self.disc = Discriminator(n\_h)

def forward(self, seq1, seq2, adj, sparse, msk, samp\_bias1, samp\_bias2):

# 正采样的hi

h\_1 = self.gcn(seq1, adj, sparse)

# 获取全局摘要

c = self.read(h\_1, msk)

c = self.sigm(c)

# 负采样的hi

h\_2 = self.gcn(seq2, adj, sparse)

# 投入判别器中

ret = self.disc(c, h\_1, h\_2, samp\_bias1, samp\_bias2)

return ret

# Detach the return variables

def embed(self, seq, adj, sparse, msk):

h\_1 = self.gcn(seq, adj, sparse)

c = self.read(h\_1, msk)

return h\_1.detach(), c.detach()